



SVI
Super-VIOR Interface Routines

Dean Alleva
Development and Evaluation Group
RD/Computing
Fermi National Accelerator Laboratory
P.O. Box 500, Batavia, Illinois 60510

October 21, 1987



~~CONFIDENTIAL~~
~~CONFIDENTIAL~~

SVI
Super-VIOR Interface Routines

-Version-
Software:1.0
Document:1.0

October 21, 1987

Dean Alleva
Development and Evaluation Group
RD/Computing
Fermilab

TABLE OF CONTENTS

| | | |
|--------|---|----|
| 1 | INTRODUCTION | 3 |
| 2 | SUPER VIOIR REGISTERS | 4 |
| 2.1 | MC68450 Registers | 4 |
| 2.2 | Board Registers | 5 |
| 3 | CHANNEL INITIATION AND CHAINING | 6 |
| 4 | POLLING AND INTERRUPTS | 8 |
| 5 | STATUS REPORTING | 9 |
| 6 | FRONT PANEL INTERRUPTS | 10 |
| 7 | CONFIGURATION ROUTINES | 11 |
| 7.1 | SVI Initialization | 11 |
| 7.2 | Setting the MC68450 Registers | 12 |
| 7.2.1 | Setting the Device Control Register | 12 |
| 7.2.2 | Setting the Operation Register | 12 |
| 7.2.3 | Setting the Sequence Control Register | 12 |
| 7.2.4 | Setting the Memory Transfer Counter | 12 |
| 7.2.5 | Setting the Memory Address Register | 13 |
| 7.2.6 | Setting the Device Address Register | 13 |
| 7.2.7 | Setting the Base Transfer Counter | 13 |
| 7.2.8 | Setting the Base Address Register | 13 |
| 7.2.9 | Setting the Normal Interrupt Register | 14 |
| 7.2.10 | Setting the Error Interrupt Register | 14 |
| 7.2.11 | Setting the Memory Function | 14 |
| 7.2.12 | Setting the Device Function Code | 14 |
| 7.2.13 | Setting the Base Function Register | 15 |
| 7.2.14 | Setting the Channel Priority Register | 15 |
| 7.3 | Channel Configuration | 16 |
| 7.4 | Interrupt Configuration | 17 |
| 7.5 | Operation Configuration | 18 |
| 7.6 | Transfer Configuration | 19 |
| 7.7 | Setting the Front Panel Mask | 20 |
| 7.8 | Clearing the CSR | 21 |
| 7.9 | Creating Command Arrays | 22 |
| 7.10 | Creating Command Chains | 22 |
| 8.0 | OPERATION CONTROL ROUTINES | 24 |
| 8.1 | Starting a Channel | 24 |
| 8.2 | Waiting For Completion | 25 |
| 8.2.1 | Waiting With Polling | 25 |
| 8.2.2 | Waiting With Interrupts | 25 |
| 8.3 | Aborting an Operation | 26 |
| 9 | STATUS REPRORTING | 27 |
| 10 | COMMENTS ON THE MC68450..... | 28 |
| | REFERENCE | 29 |

1 INTRODUCTION

The document describes a set of routines for a VME DMA module called the Super-VIOR [1]. The Super-VIOR interface routines, also called the SVI routines, are written in PILS and run under a Valet-plus system [2]. These routines enable a program to set up, execute, and monitor DMA operations. It is assumed that the reader is familiar with VME, the MC68450, and has some knowledge of the interface's design [1]. A copy of the SVI software is available on BitNet at Fermilab in "FNAL::USR\$ROOT:[ALLEVA.PUBLIC]".

The Super-VIOR Interface Routines are written in PILS, a high level language similar to BASIC and Pascal which is powerful and fast enough for most applications. One of the most powerful features of the Valet/PILS system is the ability to set up exception vectors and exception handlers directly in a program. This feature is used to handle interrupts from the MC68450 and the interface's front panel.

This document is divided into ten sections, the first being the introduction. The remaining sections detail the interface registers, channel initiation, polling and interrupts, status reporting, front panel interrupts, the configuration routines, the operation control routines, the status reporting routines, and special comments on the MC68450.

2 SUPER-VIOR REGISTERS

There are two sets of registers in the Super-VIOP DMA Interface. The MC68450 registers configure the DMA controller while the board CSR register configure front panel operations.

2.1 MC68450 Registers

The MC68450 is a 4-channel 16-bit DMA controller. Each channel has its own set of control registers. The controller also contains a single general control register which affects all channels. These registers are briefly described below. For a detailed description of these registers see the document describing the MC68450 [3].

- Device Control Register: This register configures the device oriented information for the channel. Thus, such information as port size can be set with this register.
- Operation Control Register: This register configures operation specific functions for the channel. Such information as transfer direction, operand size, and operation chaining are set in this register.
- Sequence Control Register: This register defines the count sequence of the memory and device address registers during a transfer.
- Channel Control Register: This register is used to start or terminate the operation of a channel as well as for setting interrupt request generation.
- Channel Status Register: Used to monitor the channel with polling as well as for indicating channel status upon completion of an operation.
- Channel Error Register: Contains a status code for the given channel upon operation completion. This value is only checked if the error bit in the status register is set (or an error interrupt was generated). Clearing the status register also clears this register.
- Channel Priority Register: Defines the priority level for the channel. Level 0 is the highest and level 3 is the lowest priority.
- General Control Register: There is only one general control register per MC68450. This register is used to set the burst time and bandwidth ratio for the DMA on the system bus (VME).

Address Registers: These three 32-bit registers contain the memory address, device address, and base address used during a transfer. Address count sequences for the memory and device address registers are set in the sequence control register.

Function Code Registers: These register sets the memory function codes used with memory, device and base addresses. These codes are mapped into standard VME address modifiers. The function code mapping is shown below. The lower three bits in this register map to the FCx lines (bit 0 = FC0, bit1 = FC1, bit2 = FC2).

| | | | | | |
|-----|-----|-----|-----|-----|-----|
| AM5 | AM4 | AM3 | AM2 | AM1 | AM0 |
| 1 | X | 1 | FC2 | FC1 | FC0 |

X : Depends on transfer word size. If 16-bit X equals 1, if 8-bit X equals 0.

Transfer Count Registers: These two 16-bit registers are counters use to keep track of words transfered and command array lengths.

Interrupt Registers: These two 8-bit registers contain interrupt vectors, one register for normal completion interrupt and one register for error interrupt.

2.2 Board Registers

A control and status register is present on the Super-VIOR which is used to set up front panel interrupts and indicate status of lines from the panel. For a full description of this register, see the document describing the Super-VIOR module [1].

3 CHANNEL INITIATION AND CHAINING

The Super-VIOP is a register based DMA with the option to chain commands together. These command chains are loaded by the DMA using the VME master interface. Direct manipulation of the channel registers is also possible using the configuration routines.

The SVI routines supply functions which set each registers individually as well as routines which set common configuration parameters in one call. How and when these routines are called is up to the user but a channel should be fully configured before it is started. SVI must also be initialized, with a call to `svi_init`, before any call to other SVI routines.

The ability to create chains of commands enables several operations to take place without CPU intervention. There are two types of chains available for linking commands, array and linked list chains. In array chains the blocks of parameters follow each other sequentially in memory. In linked list chains each parameter block also contains the address of the next block. Thus, linked list chains can have their parameters blocks scattered in memory.

A command chain is created before the channel which is to execute it is started. In the case of array chains, the base address of the chain is placed in the base address register and the length of the chain is placed in the base transfer counter register. In the case of linked list chains, the base address of the chain is placed in the base address register. The end of the linked list command chain is marked with zeros in the link address field of the last parameter block.

The examples below illustrate both the creation of array chains and that of linked list chains. Note that all addresses must be supplied to the SVI routines by the control program since Valet-plus has no memory management capability.

Array Chains-

```
svi_start_array (array_badd, 5, 1) ! Creates an empty array of five
                                   ! parameter blocks starting at
                                   ! address array_badd.
svi_set_entry (1, add1, 5)         ! Sets the memory transfer count
                                   ! to five and the memory
                                   ! transfer address to add1.
....
svi_set_entry (5, add2, 300)       ! Similar to entry one but sets
                                   ! entry five with different
                                   ! paramters. Note that parameter
                                   ! blocks need not be set in order
                                   ! (from 1 to 5 for this example)
                                   ! but can be initialized in any
                                   ! order as long as all blocks
                                   ! are set.
```

[illegible]

Linked List Chains-

```

svi_start_link (0) ! Start a new link
svi_add_link (link_add1, add1, 5) ! Adds a link to the present chain.
! The link is created at link_add1
! with a memory transfer address of
! add1 and a transfer count of 5.
svi_ad_link (link_add2, add2, 300) ! Adds another link at the end
! of the chain. The link is created
! at a different address and has
! different parameters.
...
svi_set_badd (ch_num, link_add1) ! Set the base address register to
! the address of the first link in
! the list.
...
svi_go (ch_num, ...) ! After all parameters are set,
! the channel may be started.

```

4 POLLING AND INTERRUPTS

Two methods are used to monitor an operation. In polling mode, the channel status register value is fetched in a loop and two bits are checked. The Operation Complete bit is set when the channel operation is completed. The Error bit is set if an error occurred during execution and indicates that the error register contains the error code. The error register is checked only if the Error bit is set. Polling mode can be used to monitor only a single executing channel.

When interrupt mode is used to monitor an operation, the first four bits are set in a flag depending upon which channels are started. When all the flag bits are cleared by the interrupt handlers, all operations have been completed. Note that a new operation can not be started on a channel whose flag bit is not yet cleared. This mode of monitoring can be used with up to four channels simultaneously.

5 STATUS REPORTING

The channel error code is returned from the polling wait routine for the polled channel once the channel operation is complete. An error code of 0 indicates no error. A program may also directly fetch status for a selected channel with a call to `svi_pol_status`.

For interrupts, the interrupt wait routine returns status for each channel. A zero (0) status indicates normal completion while a status value of negative one (-1) indicates that the channel was never started.

Status messages can be sent to the display device by setting a display parameter passed to the wait routines. The display codes are shown below.

Status Display Codes

- 0 - Display no status messages
- 1 - Display all status messages (including success)
- 2 - Display error status messages only

6 FRONT PANEL INTERRUPTS

Front panel interrupts are generated by the test signals on the front panel connectors. A mask can be loaded into the board CSR to enable such interrupts and to select which signals cause an interrupt. The board CSR can also be read to see what values the various signals have.

The SVI routines supply no interrupt handler for front panel interrupts since the front panel function is application specific. The front panel routine must be supplied by the user and linked to the front panel interrupt vector before such an interrupt is enabled. The standard VALET/Pils interrupt vector routines are used to link the front panel interrupt to a user routine. This link is shown below and should be included in any program which use front panel interrupts.

```
vcInk (16%91, 13, ADDRESS(user_routine_name))
```

The interrupt vector is hex 91, logic unit number is 13, and `user_routine_name` is the subroutine name of the front panel interrupt handler. Note that the interrupt vector and logic unit number shown above must be used for the link.

7 CONFIGURATION ROUTINES

These routines are used to set registers in the DMA controller, initialize the SVI routines, create command chains, and set the front panel mask.

7.1 SVI Initialization

1) svi_init (vbase, gcr_value)

Description: Initializes the SVI internal data structures and the 8-bit general control register. Should be called once before any calls to other SVI routines.

Parameters:

vbase (INT32, input): VME address where Super-VIOR address space starts.

gcr_value (INT32, input): General Control Register value.

7.2 Setting The MC68450 Registers

7.2.1 Setting The Device Control Register -

1) svi_set_device (chan_num, dcr_value)

Description: Sets the 8-bit device control register for the selected channel.

Parameters:

chan_num (INT32, input): the channel number (0-3).
dcr_value (INT32, input): device control register value.

7.2.2 Setting The Operation Register -

1) svi_set_oper (chan_num, ocr_value)

Description: Sets the 8-bit operation control register for the selected channel.

Parameters:

chan_num (INT32, input): the channel number (0-3).
ocr_value (INT32, input): operation control register value.

7.2.3 Setting The Sequence Control Register -

1) svi_set_seq (chan_num, scr_value)

Description: This routine sets the 8-bit sequence control register for the selected channel.

Parameters:

chan_num (INT32, input): the channel number (0-3).
scr_value (INT32, input): sequence control register value.

7.2.4 Setting The Memory Transfer Counter -

1) svi_set_mcount (chan_num, mtc_value)

Description: This routine sets the 16-bit memory transfer counter register for the selected channel.

Parameters:

chan_num (INT32, input): the channel number (0-3)
mtc_value (INT32, input): memory transfer register value.

7.2.5 Setting The Memory Address Register -

1) svi_set_madd (chan_num, mar_value)

Description: This routine sets the 32-bit memory address register for the selected channel.

Parameters:

chan_num (INT32, input): the channel number (0-3)
mar_value (INT32, input): memory address register value.

7.2.6 Setting The Device Address Register -

1) svi_set_dadd (chan_num, dar_value)

Description: This routine sets the 32-bit device address register for the selected channel.

Parameters:

chan_num (INT32, input): the channel number (0-3).
dar_value (INT32, input): device address register value.

7.2.7 Setting The Base Transfer Counter -

1) svi_set_bcount (chan_num, btc_value)

Description: Sets the 16-bit base transfer counter for the selected channel.

Parameters:

chan_num (INT32, input): the channel number (0-3).
btc_value (INT32, input): base transfer counter value.

7.2.8 Setting The Base Address Register -

1) svi_set_badd (chan_num, bar_value)

Description: Sets the 32-bit base address register for the selected channel.

Parameters:

chan_num (INT32, input): the channel number (0-3).
bar_value (INT32, input): base address register value.

7.2.9 Setting The Normal Interrupt Register -

1) svi_set_inorm (chan_num, niv_value)

Description: Sets the 8-bit normal interrupt vector register for the selected channel.

Parameters:

chan_num (INT32, input): the channel number (0-3)
niv_value (INT32, input): normal interrupt vector.

7.2.10 Setting Error Interrupt Register -

1) svi_set_ierr (chan_num, eiv_value)

Description: Sets the 8-bit error interrupt vector register for the selected channel.

Parameters:

chan_num (INT32, input): the channel number (0-3).
eiv_value (INT32, input): error interrupt vector.

7.2.11 Setting The Memory Function -

1) svi_set_mfunc (chan_num, mfc_value)

Description: This routine sets the 8-bit memory function code register for the selected channel. These codes are mapped into the standard VME address modifiers (see section 2.1). Note that only the lower three bits of this register are used.

Parameters:

chan_num (INT32, input): the channel number (0-3).
mfc_value (INT32, input): memory function code.

7.2.12 Setting The Device Function Code -

1) svi_set_dfunc (chan_num, dfc_value)

Description: Same as svi_set_mfunc but sets the 8-bit device function code register.

Parameters:

chan_num (INT32, input): the channel number (0-3)
dfc_value (INT32, input): memory function code register.

7.2.13 Setting The Base Function Code -

1) svi_set_bfunc (chan_num, bfc_value)

Description: Same as svi_set_mfunc but sets the 8-bit base function code register.

Parameters:

chan_num (INT32, input): the channel number (0-3).
bfc_value (INT32, input): base function code value.

7.2.14 Setting The Channel Priority Register -

1) svi_set_pri (chan_num, cpr_value)

Description: This routine sets the 8-bit channel priority register for the selected channel.

Parameters:

chan_num (INT32, input): the channel number (0-3).
cpr_value (INT32, input): priority value.

7.3 Channel Configuration

1) svi_chn_config (chan_num, dev, pri, mfnc, bfnc, dfnc)

Description: Sets a channel's basic information which is not likely to change often. This routine is equivalent to calls to svi_set_device, svi_set_pri, svi_set_mfunc, svi_set_bfunc, and svi_set_dfunc, see the description of these routines for more information. This routine also sets the interrupt register values automatically.

Parameters:

| | | |
|----------|-----------------|-------------------------------|
| chan_num | (INT32, input): | the channel number (0-3) |
| dev | (INT32, input): | device control register value |
| pri | (INT32, input): | channel priority |
| mfnc | (INT32, input): | memory function code |
| bfnc | (INT32, input): | base function code |
| dfnc | (INT32, input): | device function code |

7.4 Interrupt Configuration

1) svi_set_interrupts (chan_num)

Description: Sets the selected channel's normal and error interrupt registers. The values placed in the registers are internal to SVI.

Parameters:

chan_num (INT32, input): the channel number (0-3)

7.5 Operation Configuration

1) svi_set_operation (chan_num, oper, seq)

....Description: Sets operation specific information for the given channel.
This routine is equivalent to calls to svi_set_operation
and svi_set_seq, see the description of theses routines
for more information.

Parameters:

chan_num (INT32, input): the channel number (0-3).
oper (INT32, input): operation control register value.
seq (INT32, input): sequence control register value.

7.6 Transfer Configuration

1) svi_set_transfer (chan_num, mcount, madd, bcount, badd, dadd)

Description: Sets transfer specific information for the selected channel. This routine is equivalent to calls to svi_set_mcount, svi_set_madd, svi_set_bcount, svi_set_badd, and svi_set_dadd, see the descriptions of these routines for more information.

Parameters:

| | | |
|----------|-----------------|--------------------------------|
| chan_num | (INT32, input): | the channel number (0-3). |
| mcount | (INT32, input): | memory transfer counter value. |
| madd | (INT32, input): | memory address register value. |
| bcount | (INT32, input): | base transfer counter value. |
| badd | (INT32, input): | base address register value. |
| dadd | (INT32, input): | device address register value. |

7.7 Setting The Front Panel Mask

1) svi_panel_mask (mask_value)

Description: Loads a value into the board CSR [2].

Parameters:

mask_value (INT32, input): the mask value.

7.8 Clearing The Channel CSR

1) svi_clear_csr (chan_num)

- Description: Clears the CSR register for the selected channel. This operation is automatically done by svi_wait and svi_pol wait.

Parameters:

chan_num (INT32, input): the channel number (0-3).

7.9 Creating Command Array Chains

1) `svi_start_array (start_address, size, clear_array)`

---Description: Begins the creation of a parameter array. Calls to svi_set_entry sets the entries in this new array until another call to this routine. This routine can be called with the address of an old array and the fields of that array can be changed with svi_set_entry. Note that memory to memory transfers using the device address register cannot be chained, see section ten (10) for more details.

Parameters:

chan_num (INT32, input): the channel number (0-3).
size (INT32, input): number of entries in array.
clear_array (INT32, input): If set to 1 array is zeroed.

2) svi_set_entry (entry_num, mem_field, tcount_field)

Description: Sets the fields of an entry in the presently active array. Note that entries need not be set in any order as long as all entries are set before the array is used by the DMA.

Parameters:

entry_num (INT32, input): number of the entry to set,
must be in the range
0 to array size - 1.

7.10 Creating Command Chains

```
1) svi_start_link (last link)
```

Description: Starts the creation of a parameter chain. Any subsequent calls to svi_add_link adds a new parameter link to the end of the list until another call to this routine makes another chain the active chain. Old chains may be added to by supplying this routine with the address of their last link. Note that memory to memory transfers using the device address register can not be chained, see section ten (10) for more details.

Parameters:

last_link (INT32, input): if set to zero, a new chain is created. If set to the address of the last link in a list, that list can be extended.

2) svi_add_link (link_add, mem_field, tcount_field)

Description: Adds a parameter link to the end of the present list.

Parameters:

| | | |
|--------------|-----------------|--|
| link_add | (INT32, input): | VME memory address where parameter list is created. |
| mem_field | (INT32, input): | memory transfer register value. |
| tcount_field | (INT32, input): | memory transfer counter value. |

8 OPERATION CONTROL ROUTINES

These routines initiate, monitor, and abort channel operations.

8.1 Starting A Channel

1) svi_go (chan_num, int_enable)

Description: This routine starts a DMA channel. Note that the MC68450 does not allow more than one channel to be started at a time.

Parameters:

chan_num (INT32, input): the channel number (0-3)
int_enable (INT32, input): if set to 1, the channel
is enabled to use interrupts.
Otherwise, polling must be used
to monitor the channel.

8.2 Waiting For Completion

There are two ways to wait for completion of a channel or channels, with polling or with interrupts. If any channel is started with `svi_go` and the `int_enable` parameter set to 1, interrupt wait should be used. If more than one channel is to execute at any given time, interrupts should be used. Polling can be used to monitor only one channel at a time.

8.2.1 Waiting With Polling -

1) `svi_pol_wait (chan_num, display, status)`

Description: Does polling on the selected channel's status register until operation is complete. Should be used when only a single channel is to be monitored.

Parameters:

`chan_num` (INT32, input): The channel number of the channel to poll.
`display` (INT32, input): Status display code
 0 - display no status messages
 1 - display all status messages
 2 - display only error messages
`status` (INT32, output): returned status code for polled channel.

8.2.2 Waiting With Interrupts -

1) `svi_wait (display, stat0, stat1, stat2, stat3)`

Description: This routine returns the status of all started channels upon completion of their operation if they were started with `svi_go` and the `int_enable` parameter set to 1. Unlike polling, interrupts can monitor more than one channel at a time.

Parameters:

`display` (INT32, input): Status display code
 0 - display no status messages
 1 - display all status messages
 2 - display only error messages
`stat0-3` (INT32, output): Status codes for each channel.
 Returns -1 if channel was never started.

8.3 Aborting A Channel Operation

1) svi_abort (chan_num)

Description: Aborts a channel operation. This routine is useful in front panel interrupt handlers.

Parameters:

chan_num (INT32, input): The number of the channel to abort (0-3).

9 STATUS REPORTING

Status is returned to a program from the wait routines but sometimes it may be necessary to fetch status directly. The routine below allows a program to fetch the present status of any channel. Note that the wait routines clear the error code register after the codes are fetched.

1) svi_pol_status (chan_num, display, status)

Description: Returns the value of the error code register for the selected channel.

Parameters:

| | | |
|----------|------------------|--|
| chan_num | (INT32, input): | selected channel number (0-3) |
| display | (INT32, input): | display status code |
| | | 0 - display no status messages |
| | | 1 - display all status messages |
| | | 2 - display only error messages |
| status | (INT32, output): | returned status code for the selected channel. |

10 COMMENTS ON THE MC68450

- 1) It is impossible to start more than one channel at a time. Each channel has its own control register and start bit and must be started separately. It would be better if the start bits for all channels were located in a single register, like the general control register, where the channels can be started with a single access.
- 2) It is impossible to do chained memory to memory transfers. Memory to memory transfers use the device address register for one address in the transfer. Command chains contain no field for setting the device address register so that it is impossible to do unsupervised memory to memory transfers.
- 3) The MC68450 would function better if it supplied a service request completed line to indicate the completion of an interrupt request. At present, clearing the status register clears the interrupt request line but only if no other interrupts are pending. The VME interrupter can not tell when one interrupt request ends and another begins if more than one interrupt is pending.

REFERENCE

- [1] SUPER-VI0R, VMEbus Dual 16-bit Input Output register with full DMA, hardware description, Opifex AB publication (Version 1.1).
- [2] Bernes-Lee, T. et. al. The VALET-PLUS, a VMEbus Microcomputer for Physics Applications. Fifth conference on Real Time Computer Applications in Nuclear, Particle, and Plasma Physics- San Fransisco, May 1987
- [3] MC68450 Four-Channel Direct Memory Access Controller, Motorola document.